**XHR对象**

如果只支持IE7以及更高版本只需要var xhr = new XMLHttpRequest();

如要兼容更早版本或者非主流浏览器var xhr = new XMLHttpRequest() || new ActiveXObject();

1. Xhr用法

首先调用方法open（），传3个参数：请求类型‘get’或‘post’，url，和是否异步（传true则异步）。此方法启动一个请求以备发送，如果是同步的请求，js代码会等待ajax请求响应之后再继续执行。

然后send（）方法传入一个参数：发送的数据，如不发送则传null，因为这个参数对有些浏览器来说是必须的。在收到响应之后，响应的数据会自动填充XHR对象的属性。

Xhr对象有4个属性，responseText，responseXML如果内容类型是‘text/xml’或者’application/xml’，这个属性中保存响应数据的XML DOM文档，status响应的http状态码，statusText http状态码的说明。

接收到响应后第一步检查status属性，一般200算成功，responseText内容 已经就绪，此外304也成功。一般检测到200-300和304状态码算成功。有的浏览器会错误得将204状态码报错。

异步请求的时候还需要判断readyState属性，0：创建对象，1：启动，调用open，2：发送，调用send，3：接收部分，4：接收完成。每次改变值的时候都会触发readyStatechange事件。我们只需要在触发事件时检测readyState值是否等于4以及status的值就能知道请求是否完成。

此外还可以接收到响应之前调用abort（）方法来取消异步请求。如xhr.abort()。调用之后会停止触发事件以及不再允许访问任何属性。终止请求之后还应该对xhr对象进行解引用操作。

1. http头部信息

默认情况下，在发送xhr请求还会发送下列头部信息。

使用setRequestHeather（）方法设置请求头，接受2个参数：头部字段的名称和头部字段的值。且要在open之后，send之前设置

getResponseHeader方法能得到单个响应头部信息，getALLResponseHeaders方法能得到一个包含所有头部信息的长字符串。

1. 默认情况下浏览器对post请求 和web表单的请求不会一视同仁，不过可以使用xhr来模仿表单提交。设置头部‘content-Type’为‘application/x-www-form-urlencoded’,send的数据内容也需要进行序列化，见13章serialize函数。

**XMLHttpRequest 2 级**

1. FormData

现代 Web 应用中频繁使用的一项功能就是表单数据的序列化，XMLHttpRequest 2 级为此定义了FormData 类型。[Dome1](dome1.html)

1. 超时设定

12年只有IE8及以上支持超时设定。

xhr.open("get", "timeout.php", true);

xhr.timeout = 1000; //将超时设置为 1 秒钟（仅适用于 IE8+）

xhr.ontimeout = function(){

alert("Request did not return in a second.");

};

xhr.send(null);

timeout 属性设置为 1000 毫秒，意味着如果请求在 1 秒钟内还没有返回，就会自动终止。**请求终止时**，会调用 ontimeout 事件处理程序。若 readyState同时改变为 4 ，会调用 onreadystatechange 事件处理程序。可是，如果在超时终止请求之后再访问 status 属性，就会导致错误。为避免浏览器报告错误，可以将检查 status 属性的语句封装在一个 try-catch 语句当中。

1. overrideMimeType()方法

用于重写 XHR 响应的 MIME 类型。

当服务器返回的 MIME 类型是 text/plain，但数据中实际包含的是 XML。根据 MIME 类型，即使数据是 XML，responseXML 属性中仍然是 null。通过调用 overrideMimeType()方法，可以保证把响应当作 XML 而非纯文本来处理。这个例子强迫 XHR 对象将响应当作 XML 而非纯文。

var xhr = createXHR();

xhr.open("get", "text.php", true);

xhr.overrideMimeType("text/xml");

xhr.send(null);

**进度事件**

loadstart：在接收到响应数据的第一个字节时触发。

progress：在接收响应期间持续不断地触发。

load：在接收到完整的响应数据时触发。

error：在请求发生错误时触发。

abort：在因为调用 abort()方法而终止连接时触发。

loadend：在通信完成或者触发 error、abort 或 load 事件后触发。

每个请求都从触发 loadstart 事件开始，接下来是一或多个 progress 事件，然后触发 error、abort 或 load 事件中的一个

1. load事件

load 事件，用以替代 readystatechange 事件。响应接收完毕后将触发 load 事件，因此也就没有必要去检查 readyState 属性了。会接收到一个 event 对象， target 属性指向 XHR 对象实例，因而可以访问到 XHR 对象的所有方法和属性。然而，并非所有浏览器都为这个事件实现了适当的事件对象。只要浏览器接收到服务器的响应，不管其状态如何，都会触发 load 事件。而这意味着你必须要检查 status 属性，才能确定数据是否真的已经可用了。Firefox、Opera、Chrome 和 Safari 都支持 load事件。开发人员还是要像下面这样被迫使用 XHR 对象变量因为event不被全部浏览器实现。

var xhr = createXHR();

xhr.onload = function(){

if ((xhr.status >= 200 && xhr.status < 300) || xhr.status == 304){

alert(xhr.responseText);

} else {

alert("Request was unsuccessful: " + xhr.status);

}

};

xhr.open("get", "altevents.php", true);

xhr.send(null);

1. progress事件

在接受新数据期间，周期性触发。事件对象event包含3个属性：lengthComputable保存进度信息是否可用的布尔值、position已接受字节数、totalSize预期字符数。

xhr.onprogress = function(event){

var divStatus = document.getElementById("status");

if (event.lengthComputable){

divStatus.innerHTML = "Received " + event.position + " of " +

event.totalSize +" bytes";

}

};

为确保正常执行，必须在调用 open()方法之前添加 onprogress 事件处理程序（实际上所以事件最好都在open之前添加）。如果响应头部中包含Content-Length 字段，那么也可以利用此信息来计算从响应中已经接收到的数据的百分比。

**跨资源共享**

CORS（Cross-Origin Resource Sharing，跨源资源共享）是 W3C 的一个工作草案。CORS 背后的基本思想，就是使用自定义的 HTTP 头部让浏览器与服务器进行沟通，从而决定请求或响应是应该成功，还是应该失败。

1. IE对CORS的实现

微软在 IE8 中引入了 XDR（XDomainRequest）类型。能实现安全可靠的跨域通信。XDR 对象的安全机制部分实现了 W3C 的 CORS 规范。以下是 XDR 与 XHR 的一些不同之

处。

cookie 不会随请求发送，也不会随响应返回。

只能设置请求头部信息中的 Content-Type 字段。

不能访问响应头部信息。

只支持 GET 和 POST 请求。

这些变化使 CSRF（Cross-Site Request Forgery，跨站点请求伪造）和 XSS（Cross-Site Scripting，跨站点脚本）的问题得到了缓解。被请求的资源可以设置 Access-Control- Allow-Origin 头部。作为请求的一部分，Origin 头部的值表示请求的来源域，以便远程资源明确地识别 XDR 请求。

XDR 对象的使用方法与 XHR 对象非常相似。也是创建一个 XDomainRequest 的实例，调用 open()方法，再调用 send()方法。但与 XHR 对象的 open()方法不同，XDR 对象的 open()方法只接收两个参数：请求的类型和 URL。

var xdr = new XDomainRequest();

xdr.onload = function(){//请求只要响应有效就会触发 load 事件

alert(xdr.responseText);

};

xdr.open("get", "http://www.somewhere-else.com/page/");//所以的XDR对象都是异步的

xdr.send(null);

在接收到响应后，你只能访问响应的原始文本；没有办法确定响应的状态代码。而且，只要响应有效就会触发 load 事件，如果失败（包括响应中缺少 Access-Control-Allow-Origin 头部）就会触发 error 事件。但没有任何信息可用，就只能确定请求未成功了。

在请求返回前调用 abort()方法可以终止请求：

xdr.abort(); //终止请求

与 XHR 一样，XDR 对象也支持 timeout 属性以及 ontimeout 事件处理程序。

为支持 POST 请求，XDR 对象提供了 contentType 属性，用来表示发送数据的格式，如下面的例子所示。

xdr.open("post", "http://www.somewhere-else.com/page/");

xdr.contentType = "application/x-www-form-urlencoded";

xdr.send("name1=value1&name2=value2");

这个属性是通过 XDR 对象影响头部信息的唯一方式。

1. 其他浏览器对CORS的实现

Firefox 3.5+、Safari 4+、Chrome、iOS 版 Safari 和 Android 平台中的 WebKit 都通过 XMLHttpRequest对象实现了对 CORS 的原生支持。要请求位于另一个域中的资源，使用标准的 XHR 对象并在 open()方法中传入绝对 URL 即可。

与 IE 中的 XDR 对象不同，通过跨域 XHR 对象可以访问 status 和 statusText 属性，而且还支持同步请求。跨域 XHR 对象也有一些限制，但为了安全这些限制是必需的。以下就是这些限制。

不能使用 setRequestHeader()设置自定义头部。

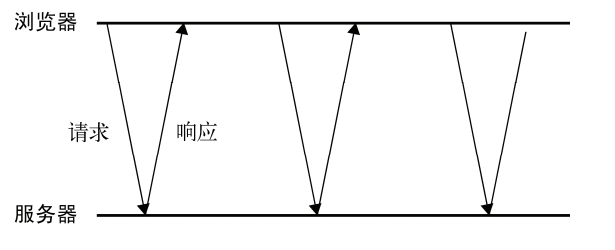
不能发送和接收 cookie。

调用 getAllResponseHeaders()方法总会返回空字符串。

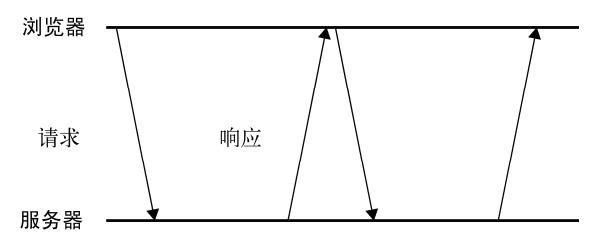
1. Comet

Comet指的是一种更高级的 Ajax 技术（经常也有人称为“服务器推送”）,是一种服务器向页面推送数据的技术。Comet 能够让信息近乎实时地被推送到页面上，非常适合处理体育比赛的分数和股票报价。

有两种实现 Comet 的方式：长轮询和流。长轮询是传统轮询（也称为短轮询）的一个翻版，即浏览器定时向服务器发送请求，看有没有更新的数据。图展示的是短轮询的时间线。



长轮询把短轮询颠倒了一下。页面发起一个到服务器的请求，然后服务器一直保持连接打开，直到有数据可发送。发送完数据之后，浏览器关闭连接，随即又发起一个到服务器的新请求。这一过程在页面打开期间一直持续不断。图展示了长轮询的时间线。



无论是短轮询还是长轮询，浏览器都要在接收数据之前，先发起对服务器的连接。轮询的优势是所有浏览器都支持，因为使用 XHR 对象和 setTimeout()就能实现。而你要做的就是决定什么时候发送请求。

第二种流行的 Comet 实现是 HTTP 流。

为它在页面的整个生命周期内只使用一个 HTTP 连接。具体来说，就是浏览器向服务器发送一个请求，而服务器保持连接打开，然后周期性地向浏览器发送数据。

<?php

$i = 0;

while(true){

//输出一些数据，然后立即刷新输出缓存

echo "Number is $i";

flush();

//等几秒钟

sleep(10);

$i+

在 Firefox、Safari、Opera 和 Chrome 中，通过侦听 readystatechange 事件及检测 readyState的值是否为 3，就可以利用 XHR 对象实现 HTTP 流。在上述这些浏览器中，随着不断从服务器接收数据，readyState 的值会周期性地变为 3。Dome3